Contents

[1. Introduction 3](#_Toc410224350)

[2. Terminology and initial research 5](#_Toc410224351)

[2.1 ETL – Extract Transform & Load 5](#_Toc410224352)

[2.2 SwidBox – ETL 7](#_Toc410224353)

[2.2.1 Orchestration bus / etcd 9](#_Toc410224354)

[3. Extracting 11](#_Toc410224355)

[3.1 Tasks and workflows 11](#_Toc410224356)

[3.1.1 Change Data Capture 11](#_Toc410224357)

[3.1.1.1 SwidBox ETL - Change Data Capture 11](#_Toc410224358)

[3.1.2 Connection with the staging service 12](#_Toc410224359)

[4. Staging 13](#_Toc410224360)

[4.1 Tasks and workflows 13](#_Toc410224361)

[4.1.1 Metadata 13](#_Toc410224362)

[4.1.2 JSON and REST 14](#_Toc410224363)

[4.1.3 Document Versioning 14](#_Toc410224364)

[4.1.4 Connections 14](#_Toc410224365)

[5. Orchestration 15](#_Toc410224366)

[5.1 Tasks and workflows 15](#_Toc410224367)

[5.1.1 Extractor Configuration 15](#_Toc410224368)

[5.1.2 Staging Configuration 15](#_Toc410224369)

[5.1.3 Task triggering 15](#_Toc410224370)

[5.1.4 Health monitoring 15](#_Toc410224371)

[5.1.5 Additional statistics 15](#_Toc410224372)

[6. Presentation 15](#_Toc410224373)

[6.1 Tasks and workflows 15](#_Toc410224374)

[6.1.1 User Management 15](#_Toc410224375)

[6.1.2 Token Management 15](#_Toc410224376)

[6.1.3 Data display 15](#_Toc410224377)

[7. Un-staging and Loading 15](#_Toc410224378)

[7.1 Tasks and workflows 15](#_Toc410224379)

[7.1.1 Loading services 15](#_Toc410224380)

# Introduction

One of the main challenges many industries are facing, is the lack of any means of controlling the database and other file related changes. Our industry makes no exception, since we have to manage probably some of the most complex applications in the world, which run on some of the most exotic systems. One area is especially challenging for all, which is the maintaining of the configuration between the many environments. Each environment consists of many files (both structured and unstructured) that are frequently being changed by different users and processes. This makes the task of tracking the state of the environments a true challenge.

As an example, a typical payment system consists in dozens of structured files (tables), and hundreds of unstructured ones (edit files and objects). All this files are then modified (updated) by users, online processes and batches. Furthermore, each environment needs to be duplicated to serve different purposes: development (DEV), unit testing (UNI), system testing (SIT), user acceptance testing (UAT) and production (one primary and one secondary), and each system is then maintained by different teams from different organizations. Furthermore there is an additional split between different clients, regions and logical networks so the effort of tracking, promoting or reverting changes grows exponentially to a point where the health of the production environments is put at risk. Maintaining different payments system types by the same teams with the same skillset seems to be taken of a science fiction movie.

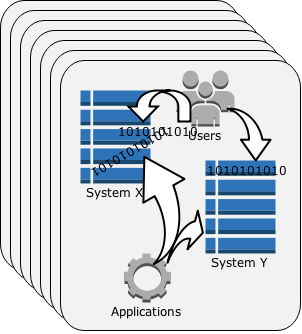


Figure 1 A typical payments system setup

This document describes the SwidBox; an open-source revolutionary distributed application that is aiming at providing a clean and solid-proof solution to the data-change headaches. The main goal of SwidBox is to enable organizations to take control over their data by providing the means of tracking changes, clearly identifying them with the option of promotion further up to the production chain, together with any transformations needed.

*some of the capabilities of SwidBox are summarized below:*

*(draft)*

* *Data warehouse-ing*
  + *Database agnostic data staging. The result after transformation is a human-friendly intermediary format (JSON) which will be stored to a specialized database as is.*
  + *A convenient and reliable way to maintain the staged data offsite. A storage area for the change-sets under supervision/control (Document - NoSQL database).*
  + *Ways of creating different reports or tools and generating business knowledge for different parts of your organization (Business analysts can see how the production is currently configured, developers can set up their sand/boxes with relevant configuration, and testers can make their test scenarios more relevant by following the UAT configuration.*
  + *Means of comparing entire environments, generate data views, and arbitrary reports*
* *Data change versioning* 
  + *When was the file/table/database changed*
  + *What were those changes about (deltas in a human-readable form)*
  + *What changes to the database have been captured between two time-frames*
  + *Revert a table or file back to specific version*
* *Data promotion (deployment) further in production line*
  + *Cherry-pick some of the changes, transform them if needed to match the target system, and promote them to a specific environment*
* *Easy to use WEB interface and adjacent web services for binding the above components together.*
  + *Task creation and monitoring*
  + *Data management*
  + *User management*
  + *Statistics*

# Terminology and initial research

In enterprise and web architectures, data systems are typically categorized into source-of-truth systems that serve as primary stores for the user and application generated writes, and derived data stores which serve reads and other complex queries. The data in these secondary stores is often derived from the primary data through custom transformations, sometimes involving complex processing driven by business logic. A fundamental requirement emerging from these kinds of data architectures is the need to reliably capture, flow and track primary data changes.

In our industry, where the source-of-through is not SQL derived, the custom transformations usually involve interpretation of large binary blocks (from structured or unstructured) files into internal application structures. In order to maintain those transformations efficiently, Domain Specific Languages (DSLs) had to be designed to ease the serialization processes (ex: HP NonStop DDL, MetaMan (ACI), Apache Avro, Google Protocol Buffers and AtomBox DDL). The DDL languages are also referred to as meta-data generators.

Most of the efforts in this area have been focused onto data replication systems such as Oracle GoldenGate, DRNet, Attunity CDC and IBM InfoSphere. Although there can be many lessons that can be learned from these approaches, they cannot solve the change-data-management challenge. Most of the replication solutions are focused on the low-latency and data throughput so they usually don’t perform complex data transformations themselves, but they pass this responsibility to the target recipients.

SwidBox does not fall into the real-time processing constraints, which opens new ways of approaching data changes such as data transformation for offsite storage. The source and destination in the proposed solution are interchangeable so specialized services will transform the captured data into an intermediary format before loading into a data store for further maintenance and versioning. This reversible process will be referred to as data staging.

## ETL – Extract Transform & Load

ETL (Extract/Transform/Load) – is an architectural design pattern used to describe the data movement and transformation processes. It is normally used by many industries to consolidate data from different homogeneous and heterogeneous sources into a common data-source format (data warehouse).

**Extract**

The first part of an ETL process involves extracting data from various source systems. The main challenge is to retrieve data from the source systems with as little resources as possible.

In to our industry this effort is topped with the additional complexity of physically accessing the data structures (Enscribe or c-tree) and capturing changes.

**Transform**

This step involves the transformation of the data extracted from the source into a unified format, common through the entire data warehouse. Also these steps usually involve data validation, integrity checking, cleansing, derivation of new values, aggregations and so on.

**Load**

This step involves loading the result of the transformation phase into a database (also called staging area) or any other landing areas.

## SwidBox – ETL

SwidBox ETL is an extension to the standard ETL methodology that considers the Source and Staging Area interchangeable.

A deployment methodology can be implemented by considering the Staging Area a possible source of data, which can then go through un-staging and subsequently loaded into a destination.

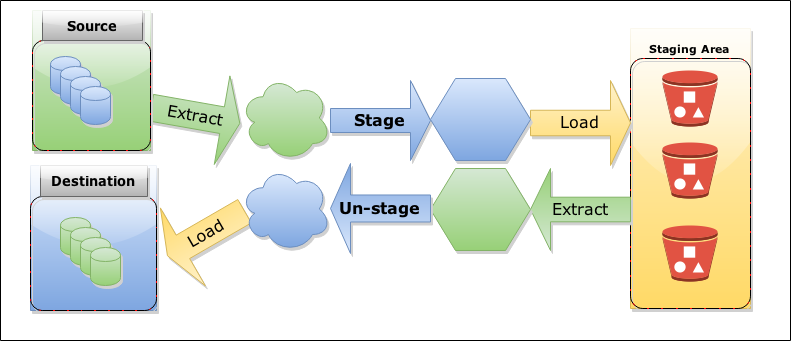


Figure 2 Extract/Transfer/Load in SwidBox

By considering the above approach, SwidBox will consist of the following types of services:

* **Source Extractors** - these services will capture the data from each source and pass it to the corresponding staging services.
* **Staging Services** - these services will retrieve the data from extractors, will identify the corresponding metadata for data transformation and then load the result after transformation into the Staging Area. As it will be described in the following sections, the Staging Services will offer support for the Source Extractors during the Change Data Capture process.
* **Staging Area** (No-SQL Database) – The output from the staging services will be in JSON (JavaScript Object Notation) format therefore the staging area will be a specialized No-SQL database that stores JSON natively called Apache CouchDB, a highly scalable, fault tolerant database that completely embraces the Web. Also as opposed to other ETL implementations, the SwidBox Staging Area is not transient in nature (does not get deleted or cleaned), and will act as both data warehouse and data source endpoint (for the deployment flow). More details on JSON and No-SQL can be found in [Section 4.1.2](#_JSON_and_REST)
* **Presentation and orchestration -** All services can all run independently on different machines or clusters. SwidBox ETL includes orchestration services, which will track and orchestrate with the rest of the services spread across the systems through the Orchestrator Bus (etcd- a distributed key/value store).
* **Un-staging Services –** these services will be triggered by an orchestration service (through the orchestrator/etcd) to extract data from the Staging Area, transform it using the corresponding meta-data to the destination format, and pass it to the destination loader for applying it to the target.
* **Destination Loading Services –** communicate with the un-staging services to apply the physical changes into targets.

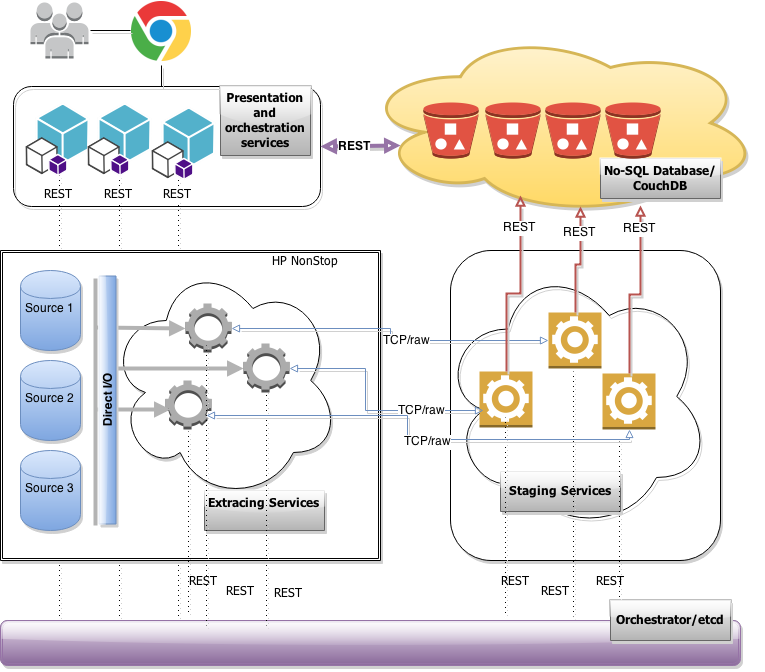


Figure 3: SwidBox – staging - high level architecture

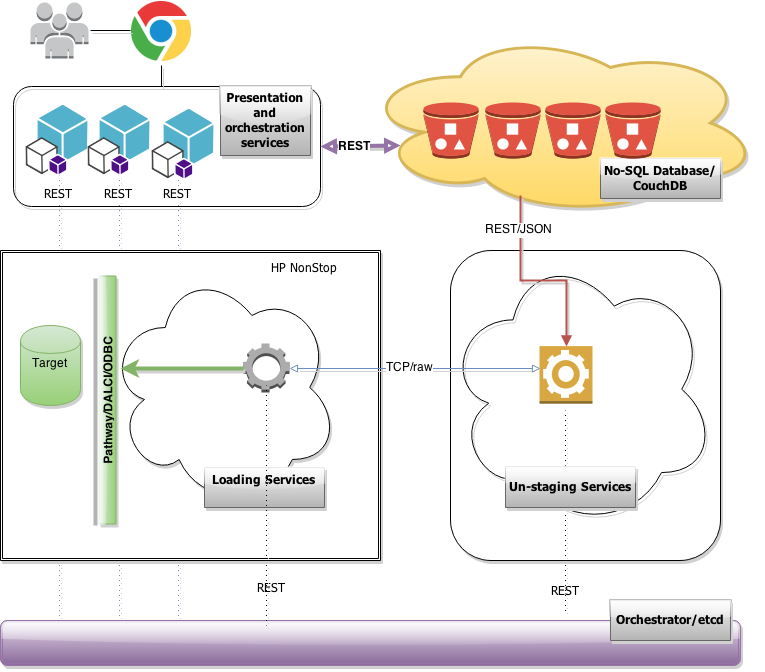


Figure 4: SwidBox – un-staging - high level architecture

### Orchestration bus / etcd

The orchestration bus is *etcd* - a daemon that runs across all computers in a cluster, allowing configuration data to be easily shared by the services by providing dynamic configuration registry.

The services run independently and the orchestration messages are being passed through *etcd*. Extracting Services use *etcd* as a discovery service in order to find their Staging peers, the Staging services use *etcd* to get the CouchDB connections and information on the metadata used in the staging process, and so on.

All extractors will maintain a dual connection with *etcd (REST & long-pooling)*, in order to report their state progress and health and receive for commands or new configuration from the Orchestration Service.

More information on etcd can be found at: <https://github.com/coreos/etcd>

The following sections will detail each of the artifacts and services, together with the technical implementation details.

# Extracting

As already presented in [Section 2](#_SwidBox_–_ETL), the first step involved in a SwidBox ETL is capturing any change of the data set under change control and passing it further to the corresponding Staging Agent.

## Tasks and workflows

In the following paragraphs will be going through each workflow that involves an extracting service.

### Change Data Capture

Change Data Capture (CDC) is a set of software design patterns used to determine (and track) the data that has changed so that an action can be taken using the changed data.

In SwidBox ETL, the Staging Area (CouchDB) contains a staged mirror of the data sets under version control therefore a synchronization process needs to be implemented; this represents a challenge on itself.

As described in [Section 2](#_Terminology_and_initial), There are not many pioneers in the field of CDC and the approaches are limited to either tracking changes using Database Triggers (or interceptors) or by reading the transaction log as is (ex: TMF) .

Neither of the two approaches seems to be suitable for SwidBox ETL since most of the time both an interceptor and digging the transaction log need to be implemented in the same solution. As an example HP NonStop you cannot have multiple intercept-libraries active at one point in time and also the transaction log (TMF) is not enabled for all files.

Fortunately, SwidBox does not have to follow the same constraints as the real-time replication services, since low-latency is not required in the process.

### SwidBox ETL - Change Data Capture

Where the database do not support triggers (like SQL), CDC can be achieved in a non-intrusive way by checking at regular intervals for changes based on the different types of file as follows:

#### Unstructured

For an unstructured file (object or text) – SwidBox will compute one unique 128 bit (non-cryptographic) hash value referred SWID through the rest of the document, which will be compared against the SWID retrieved from the Staging Service. If the SWID doesn’t match then it will be passed further down the chain to the Staging Agent.

#### Structured

The structured files are characterized by a positioning value (primary key) that facilitates sequential access. This primary key can be any binary data (key sequenced), record number (relative) or offset (entry sequenced).

For these types, the ‘last modified time’ is checked first, then the Agent will compute an unique 128 bit (non-cryptographic hash/SWID) for each row that will then be compared against the corresponding value returned by the Staging Agent.

For this file types, in addition to the list of swids, the Staging Agent will also returns the list of keys in order to facilitate the record positioning.

#### Structured – Append/Only

These types are normal structured files that for optimization purposes are being flagged as Append/Only (as part of the initial configuration). This special file type is introduced to facilitate the staging of large files, especially logs such as EMS. For this file type, the Agent it will check the ‘last modified time’ and then it will send all the rows from to the last key returned by the Staging Agent.

### Connection with the staging service

Although low latency and high data throughput is not a constraint in a SwidBox setup, the Extracting Services need to be non-intrusive, in order not to affect the systems under surveillance. For this reason, the CPU cycles from the source machines are saved by the peering Staging Services that will usually run on low cost commodity hardware.

The link between the Extracting Service and its corresponding Staging Service is initiated through custom binary TCP/IP protocol. The connection details including the matching details will be *discovered* from the *etcd* bus connection that both types of services are required to retain.

# Staging

This second step in the SwidBox ETL is responsible with the transformation of the (serialized) data received from the Extracting Service and loading the result into the Staging Area, a CouchDB database instance.

## Tasks and workflows

In the following paragraphs will be going through each workflow that involves a staging service.

### Metadata

Theoretically, metadata is defined as the data providing information about or more aspects of the data such us means of creation and data meaning (field’s names, structures, lengths or types).

In the payments industry the following types of metadata the most common are:

1. DDL – HP NonStop Data Definition Language
   1. Complex DSL inspired by the COBOL 85. It supports nested structures, data redefines, arrays (occurrences), enumerations, data type references, key definitions and so on.
   2. DDL is used to generate application source code to be compiled statically (C, COBOL and TAL data structures).
2. MetaMan Language– a proprietary DSL for Base24-eps product from ACI Worldwide
   1. Rudimentary and flat DSL inspired by SQL. It does not support nested structures, redefines or arrays but it supports key and record definitions.
   2. As with the HP DDL, MetaMan is used to generate application source code to be compiled statically (C structures).
3. SQL Schemas
   1. Formal language understood by Relational Databases for organization of data as a blueprint of how a database is constructed.

The DDLs above are common structures that will be used in staging so understanding these types of metadata is a key for a successful implementation of SwidBox ETL.

The immediate deduction by looking at the various types of DSL is that interpretation of all the various types of binary data without binding in the source code (generated from the DSL) from the application is a true technical challenge. Luckily, this challenge is solved by AtomBox (www.atombox.org), a cross platform scripting engine which has (endian aware) out-of-the-box capabilities of dynamically interpreting DDL, MetaMan (and SQL schemas) into JavaScript classes. This makes the generation of JSON (JavaScript Object Notation) seamlessly. Any binary data passed by the Extracting Agent can be staged out of any binary record as long as there is a metadata structure that describes it.

### JSON and REST

JSON (JavaScript Object Notation) is a lightweight data-interchange format that uses human-readable text to transmit data objects consisting of attribute-value pairs. It is based on a subset of JavaScript programming language which makes any JSON a valid JavaScript object.

REST (Representational State Transfer) is a software architectural style consisting of best practices for building scalable web services with the data being exchanged being JSON. It gained widespread acceptance across the WEB as an alternative to SOAP and WSDL.

The release of JSON together with the REST caused the XML popularity to drop dramatically including the XML based protocols (SOAP and WSDL). Nowadays XML support in the new emerging technologies is close to inexistent.

JSON was a turning point also for the future of databases the NO-SQL (JSON storing databases) taking the spotlight over the traditional relational databases with JavaScript becoming the chosen language to query databases instead of SQL.

SwidBox makes use of the latest technologies with JSON being the output from the staging phase, JavaScript (AtomBox) the development language for Extracting and Staging Services, CouchDB the JSON document storage and REST the communication protocol.

### Document Versioning

In addition to parsing the binary data received from the Extractor, the table versioning falls into the responsibilities of the Staging Service. Every change to the database tracked and staged will have an increased version number.

### Connections

Each staging agents will maintain connections with:

* Extracting peer - TCP/IP custom binary protocol
* *etcd* (the Orchestrator) - REST/JSON
* CouchDB database - REST/JSON

# Orchestration

## Tasks and workflows

### Extractor Configuration

### Staging Configuration

### Task triggering

### Health monitoring

### Additional statistics

# Presentation

## Tasks and workflows

### User Management

### Token Management

### Data display

# Un-staging and Loading

## Tasks and workflows

The un-staging process is the reversed staging where the JSON structures are then transformed back to binary and passed to the Loading Service to be deployed into the target system. See [Section 4](#_Staging) for more information on Staging Services.

### Loading services

As opposed to the Extracting Service, Loading normally does not imply accessing the target file-system directly but formatting and passing the loading requests to specialized application services which are built these exact purposes (Pathway Servers, DALCI, etc.).